General Advice

You **don't need to delete or simplify your entire system** just to match the base API. Having "an API on top of your API" is totally valid. A great analogy from Java is the classic `LinkedList` class.

A `LinkedList` implements **multiple interfaces** simultaneously:

- `List`
- `Deque`
- `Queue`

But the internal structure of a linked list doesn't magically change depending on which interface you use. It will always be a linked list underneath. What *does* change is the **view of the API** you're choosing to expose.

For example:

```
List myList = new LinkedList();
```

Now `myList` behaves like a **List**, giving you the List methods (`get`, `add`, `remove`, etc.).

But these both point to the **same underlying object**:

```
Queue myQueue = new LinkedList();
```

Here you get **Queue** semantics (`pop`, `push`, etc.), even though it's still the same linked list internally.

If you translate this to python and ABCs, you have this:

```
from abc import ABC, abstractmethod

class ListAPI(ABC):
    @abstractmethod
    def add(self, x): ...
    @abstractmethod
    def get(self, i): ...

class QueueAPI(ABC):
    @abstractmethod
    def enqueue(self, x): ...
```

```
    @abstractmethod
    def dequeue(self): ...
```

And a single class could implement both:

```
class LinkedList(ListAPI, QueueAPI):
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def get(self, i):
        return self.data[i]

    def enqueue(self, x):
        self.data.append(x)

    def dequeue(self):
        return self.data.pop(0)
```

AND CRUCIALLY notice that that there are **NOT** two arrays for the list, it just has one but different methods to access it.

Now you can treat it as whatever API you want:

```
lst: ListAPI = LinkedList()
q: QueueAPI = LinkedList()

lst.add(5)     # using list API
q.enqueue(6)   # using queue API
```

Again — same object, different surfaces.

So, your implementation can be *more detailed* or support more operations internally than what the **common API** exposes. That's **not** a conflict. The base API is the "interface" we're all guaranteeing—not a mandate to destroy the richness of your service.

You can keep your existing capabilities internally and simply **adapt them** to the shared surface API. One thing can be multiple things, just like LinkedList is a List, a Deque, and a Queue. Our implementations can still be "ticketing systems" without flattening everything.